Прикладная эзотерика

Андрей Аксёнов
Avito + Sphinx



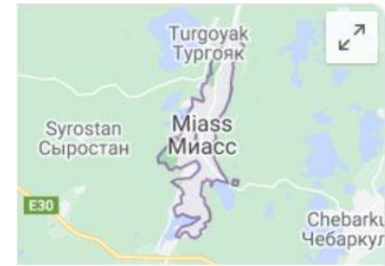




…эзотерика.

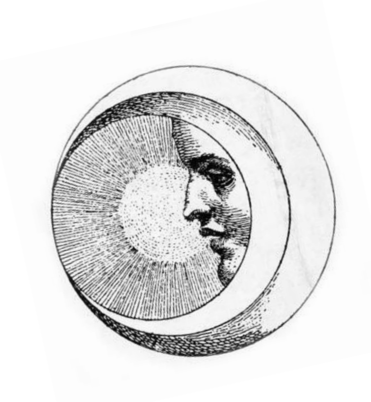


…прикладная, Miass.

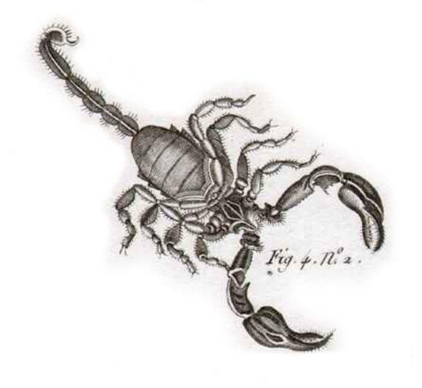








Луна в Скорпионе => билд не катим, ага.



никто. не читает. ничего.

Никто не читает **ничего**

- Вы вот читали abstract доклада? 

RhQ:

- Случались ли собеседования?
- Случались ли дурацкие вопросы “про СД”?
- И что вы на них ответили?
- И где ж вы все ходите?!

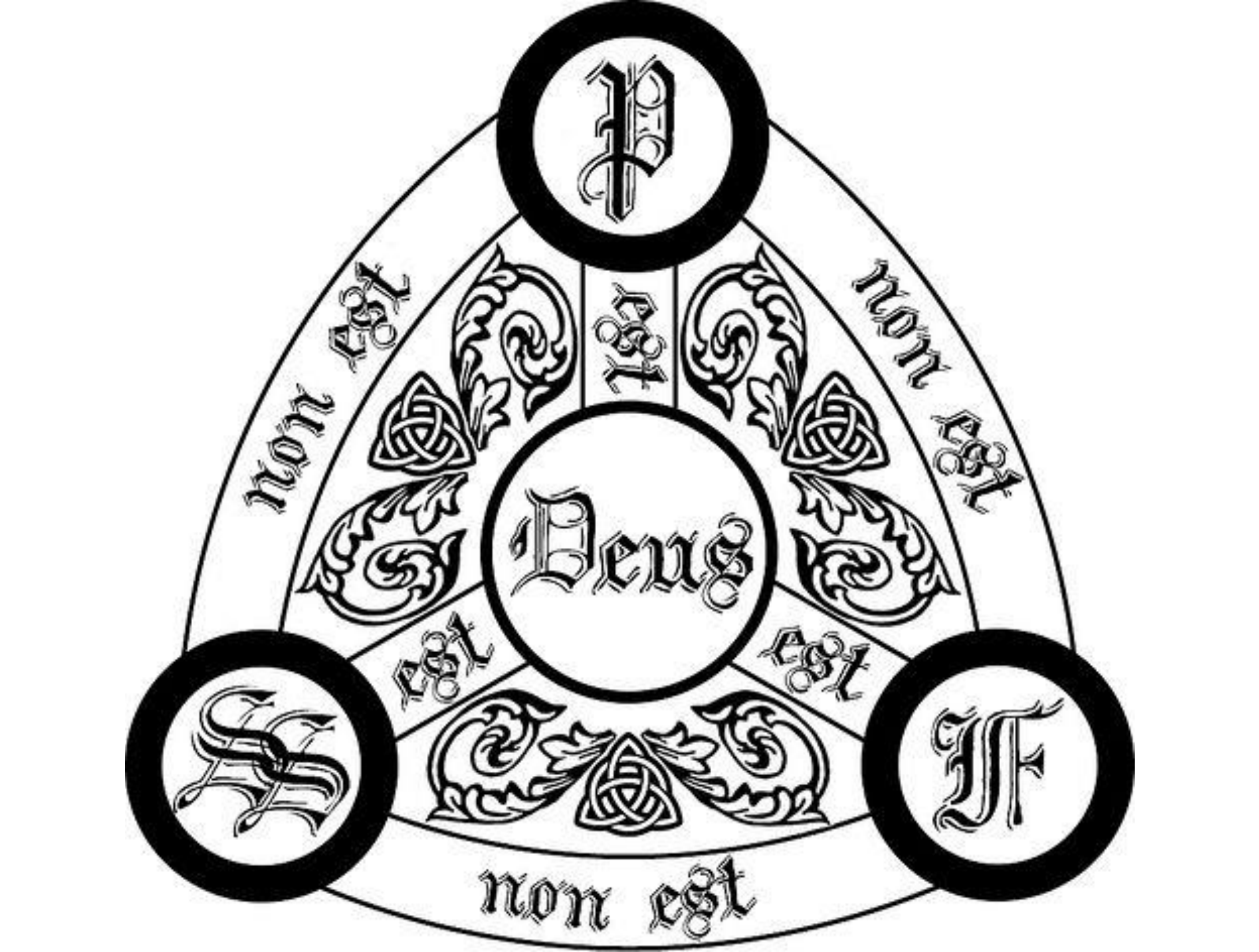


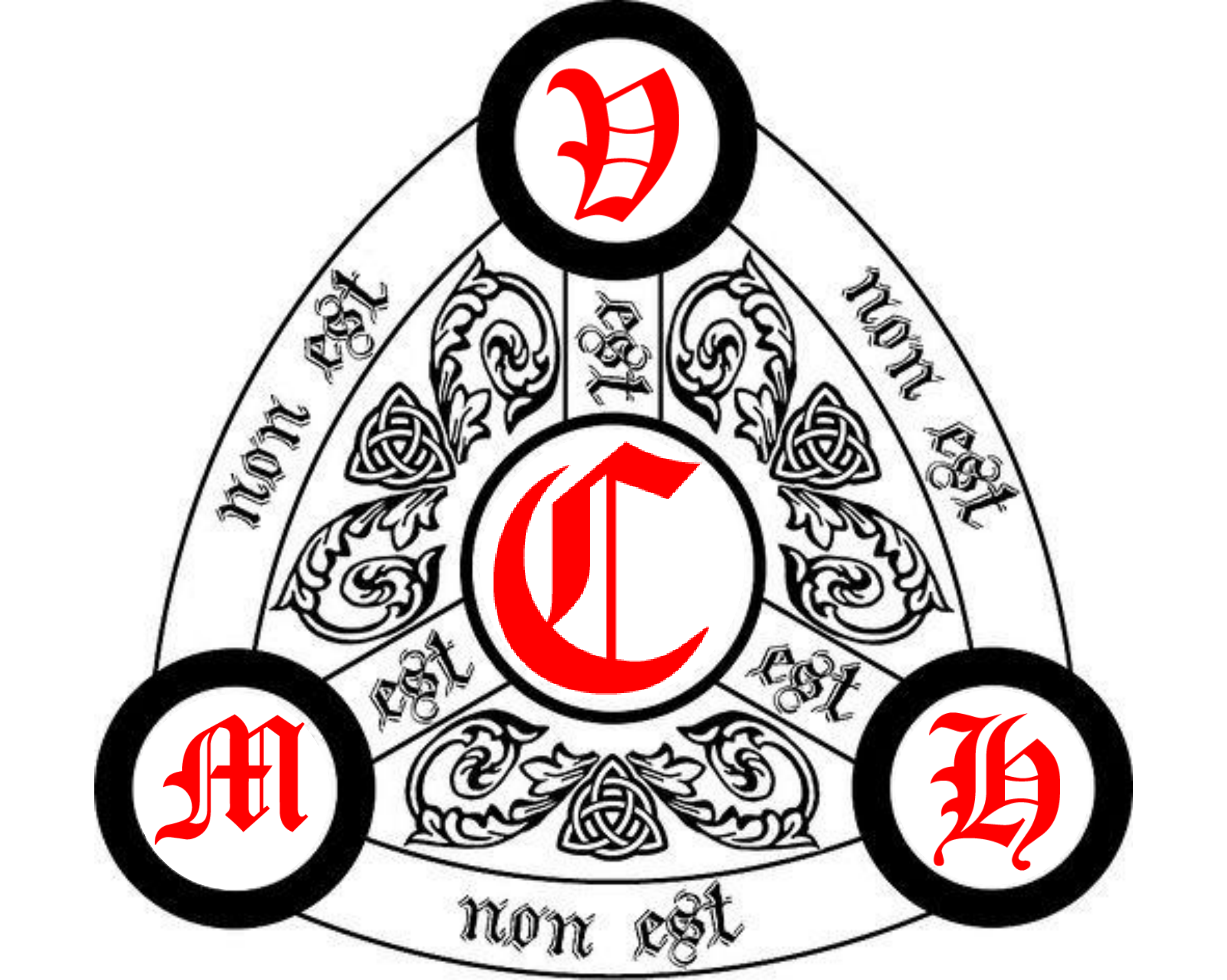
О чем **заявлен** доклад

- **Прикладная** эзотерика
- Иными словами, всё, ***кроме*** св. Троицы






P
SS
F
Deus
est
est
est
non est
non est
non est


V
C
M
H
non est
non est
non est
est
est
est

Терминологическое интермеццо

- **V** for ~~Vendetta~~ std::vector
- **M** for ~~Mandragora~~ std::map
- **H** for ~~Heresy~~ std::unordered_map

- “По-вашему” немного иначе
- .py == list, ?, dict
- .go == slice, ?, map



…Last-minute прозрение, блин

- Может, **std::map** вдруг и *есть* эзотерика?!…
  - Ну, какое-то бинарное дерево (поиска!)
  - Ну, какое-то там red-black, неважно
  - Ну, какие-то там O(log N), неинтересно
- …А что такое “эзотерика”-то?




what are other words for esoteric?


secret, abstruse, recondite, occult, arcane, deep, mysterious, mystic, cryptic, obscure


Thesaurus.plus

esoteric

adjective

UK / ˌiː.səˈter.ɪk/ **US** / ˌes.əˈter.ɪk/



very unusual and understood or liked by only a small number of people, especially those with special knowledge.

И чо?

- {vector, hash} != “esoteric”
  - да, массам ***и они*** непонятны – увы!
  - нет, они очень “обычные” (и при этом ооочень нужны)
- {fib-heap, seg-tree, ctrie, …} == “esoteric”
  - При этом ваще никому и даром не нужны!!!
- {map, …} == таки могут оказаться эзотерикой “для вас”
  - Таки при этом, однако, иногда (иногда) **нужны!**



…вот по ***нужным**** 10 шт и побежим

* – а) ну хоть иногда + б) ну хоть лично мне + в) your mileage *will* vary

…галопом; но доклад “бесконечный” ☺

разминка!



bitmap (ага; уровень “ясли”)

Q1: Как сохранить n=3…17+ шт из [0,R=1024)?

Q2: И при этом **быстро** проверять IsThere(x)?

- Влобно vector<int> vals(0); // дорастет до n
- Reset() быстрый, Add() быстрый
- IsThere() нутакое, переборрр



bitmap

Q1: Как сохранить n=3…17+ шт из [0,R=1024)?

Q2: И при этом **быстро** проверять IsThere(x)?

- Битмап vector<uint> bm(R/32); // сразу!
- IsThere() быстрый, Add() быстрый
- Reset() нутакое, заливка 1 кбит нулями



bitmap

Q1: Как сохранить n=3…17+ шт из [0,R=1024)?

Q2: И при этом **быстро** проверять IsThere(x)?

Q3: Что, если мы часто эту маску ресетим?!



SparseSet

SparseSet

```
struct SparseSet { // sketch
  int n = 0;
  int dense[R]; // uninitialized okay because n
  int sparse[R]; // uninitialized intentionally!

  void Add(int val) {
    dense[n] = val;
    sparse[val] = n;
    n++;
  }
```




SparseSet

```
struct SparseSet { // sketch
  int n = 0;
  int dense[R]; // uninitialized okay because n
  int sparse[R]; // uninitialized intentionally!

  void IsThere(int val) {
    return sparse[val] < n && dense[sparse[val]] == val;
  }

  void Reset() { n = 0; }
```






эзотерика есть?! даёшь **практику**

1. Почему работает?!
2. Зачем надо?
3. Чем плохо?
4. Где взять?



SparseSet

- Почему работает?!
  - ***заполненные*** dense/sparse “смотрят друг в друга”
  - а в незаполненные мы не ходим!

  - int X = sparse[val]; // mb garbage!
  - out-of-range мусор X не пройдет sanity check (X < n)
  - in-range мусор X не пройдет “по dense”
  - вредная задача со звездой: найдите баги :)

- …Что интересно, даже ***это*** слишком подробно!



SparseSet

- Почему работает?!
  - **“красивый логический трюк про защиту от мусора”?** :)
  - s/ло/ма …

- Когда надо?
  - Когда в среднем совсем немного Add()
  - Когда такая частая куча Reset(), что bitmap-у плохо
    - например, vars liveness sets в компиляторе?
    - например, matched field mask в FT движке?



SparseSet

- Чем плохо (vs bitmap)?
  - Память жрет в 32…64 раза, 32…64 бита вместо 1 бита!
  - Для Range=1024 будет 4…8 кб вместо 0.125 кб
  - Add() таки помедленнее bitmap
  - …короче, надо ***вдумчиво*** тестировать vs bitmap

- Где взять?
  - хз, может Folly
  - Написать!

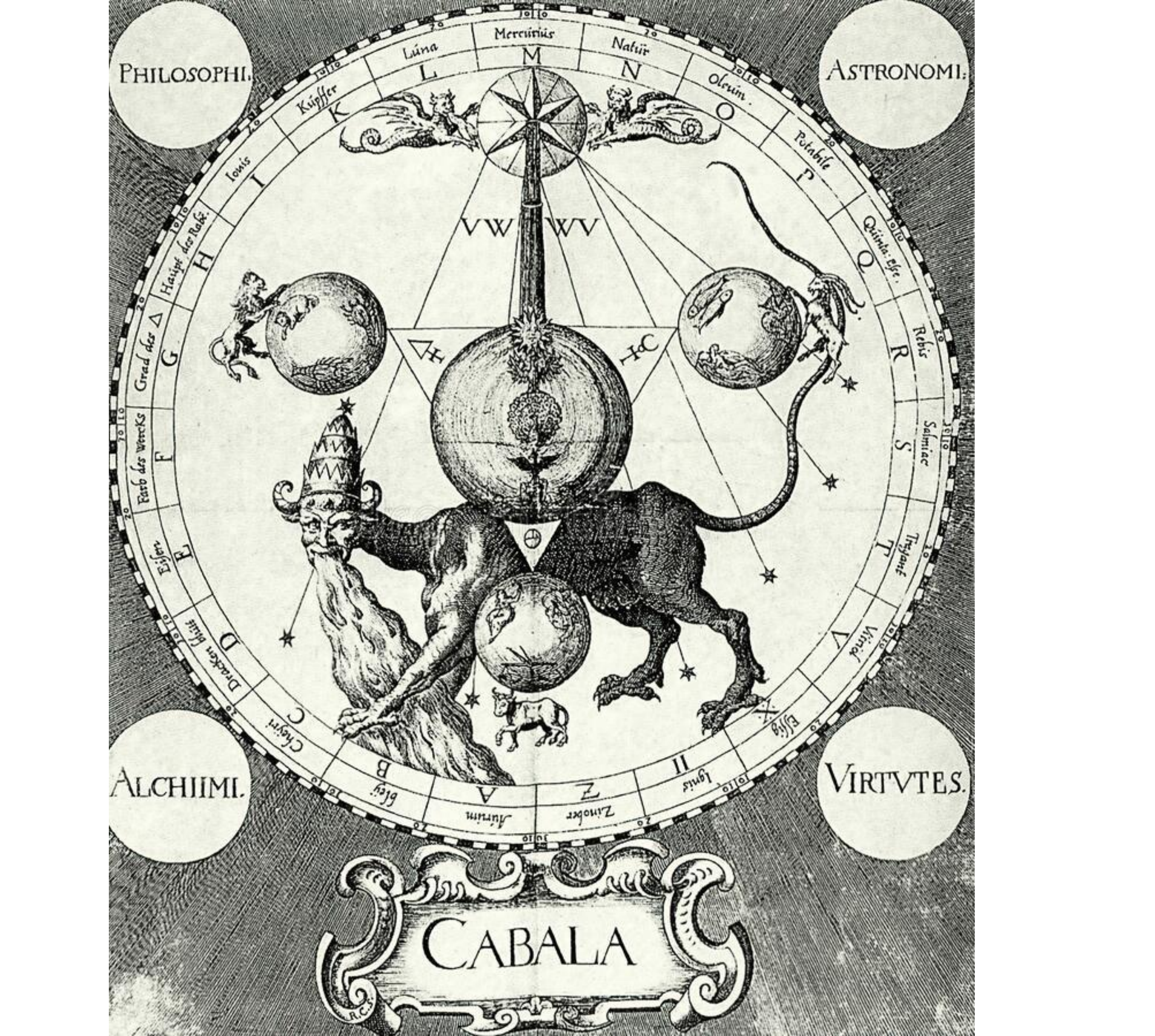




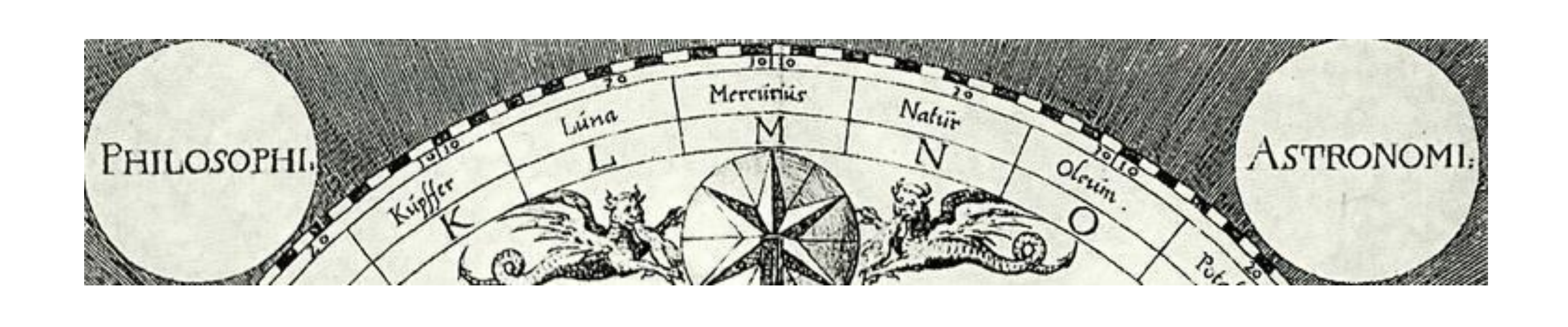
…написать? really???... ;((

разминка-2!

google “esoteric”



PHILOSOPHI.
ASTRONOMI.
ALCHIIMI.
VIRTVTES.
Mercurius
Luna
Natür
Oleum
Potabile
Rebis
Kupffer
Iovis
Eisen
Zinober
Aurum
Ignis
VW WV
CABALA


PHILOSOPHI.
ASTRONOMI:
Mercurius
Luna
Natur
Kupffer
Oleum.
K
L
M
N
O



difference-list

table

2-3-heap

b-tree

heightmap

kd-tree

splay-tree

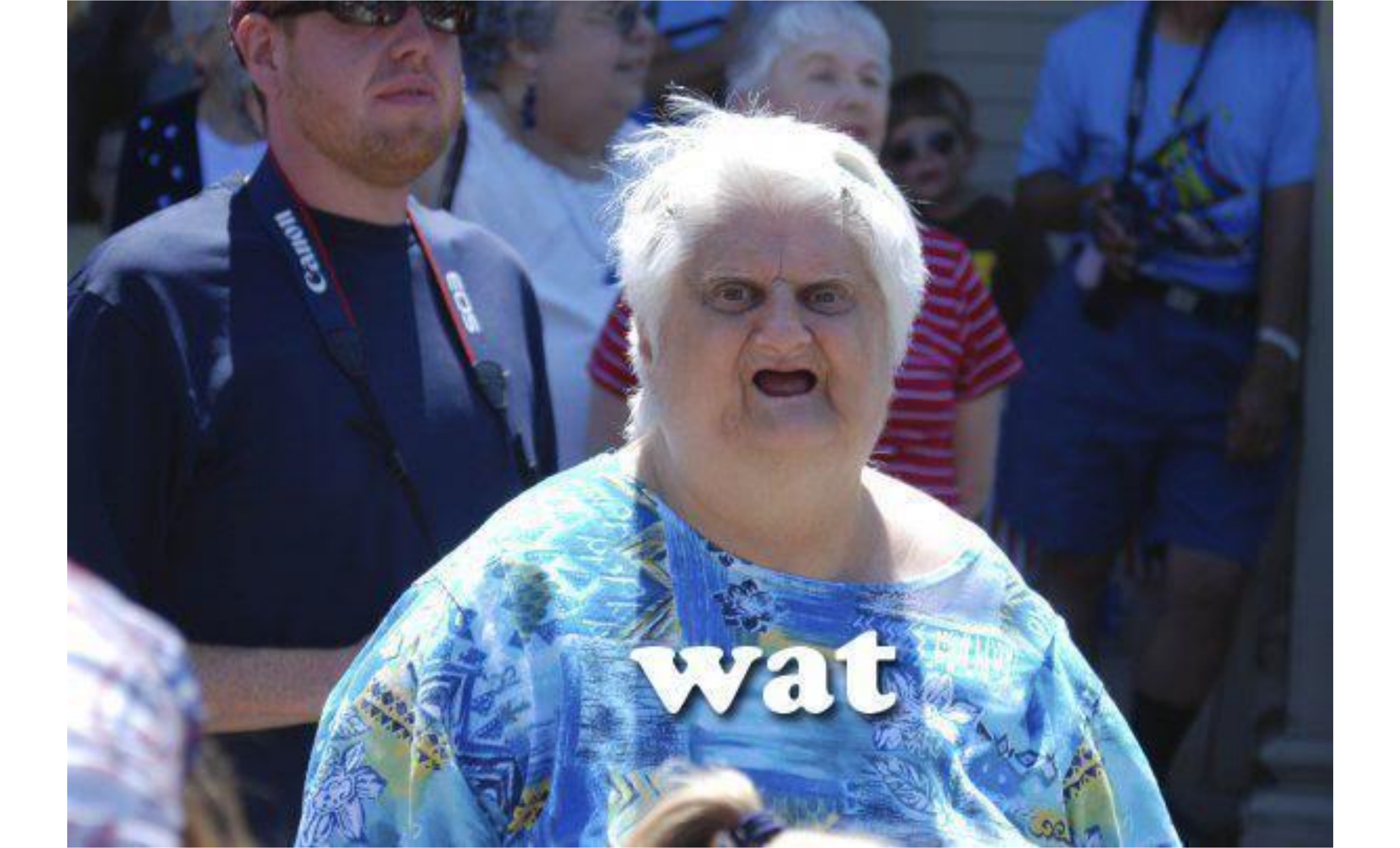
lookup

rolling-hash

skip-list

хоп, целый Стэнфорд, но внутри банально…

BloomFilter


Canon
EOS
wat

BloomFilter

Q1: хотим быстро проверять “есть ли объект”

Q2: хотим тратить *очень* мало памяти

Q3: готовы это делать немного неточно!



BloomFilter

```
struct BloomFilter { // sketch
  static const int BITS = 1024;
  Bitmap<BITS> bm; // fixed size, yes yes!

  void Add(Object & val) {
    bm.SetBit(hash1(val) % BITS);
    bm.SetBit(hash2(val) % BITS);
  }

  void IsThere(Object & val) {
    return
      bm.IsSet(hash1(val) % BITS) &&
      bm.IsSet(hash2(val) % BITS);
  }
```




закрепляем чеклист!!

1. Почему работает?!
2. Зачем надо?
3. Чем плохо?
4. Где взять?



BloomFilter

- Почему работает?!
  - Тк. шансы “попасть” во *все* нужные биты малы
  - Тк. неточный, может ошибиться, false positive
- Зачем надо?
  - **Супер-**быстро и **супер-**компактно MaybeThere()
  - Конкурентов почитай* нету
  - vector<Object> не конкурент!!! (см. **супер**-компакт)
  - vector<uint> hashes хотя бы надо (и всё равно meh)



BloomFilter

- Чем плохо?
  - Неточный (ну а шо поделать)
  - Можно промазать и будет *абсолютно* неточный
  - Можно “просто” промазать с размером, nfuncs, итп
    - Ну а ты не мажь!!!
- Где взять?
  - Вероятно, написать!
  - Srsly, погуглил 2 мин, норм “коробок” не нашёл



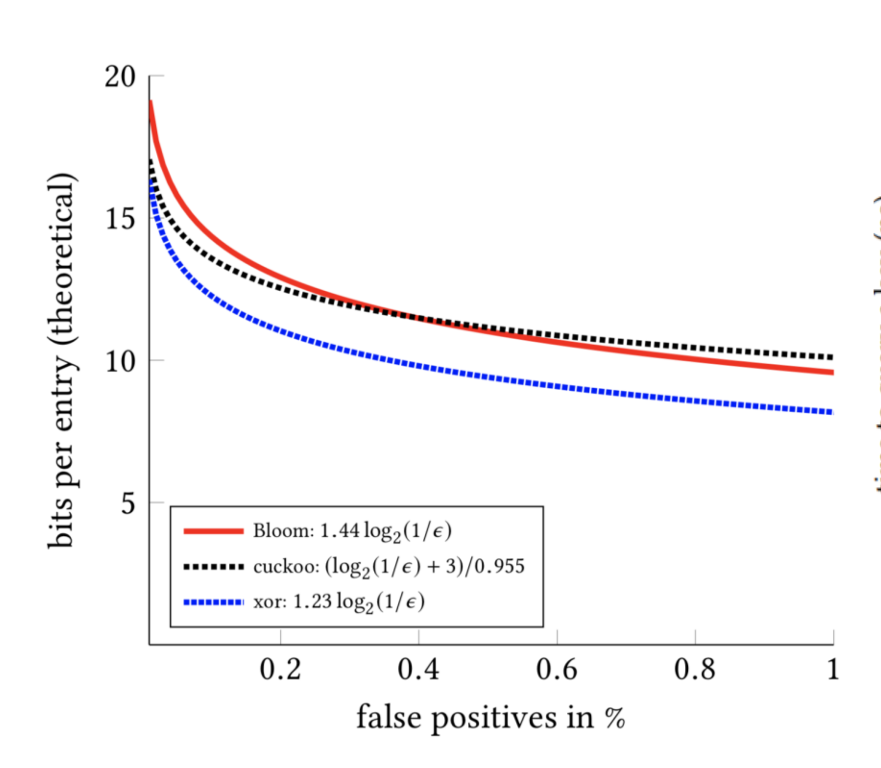
XORFilter

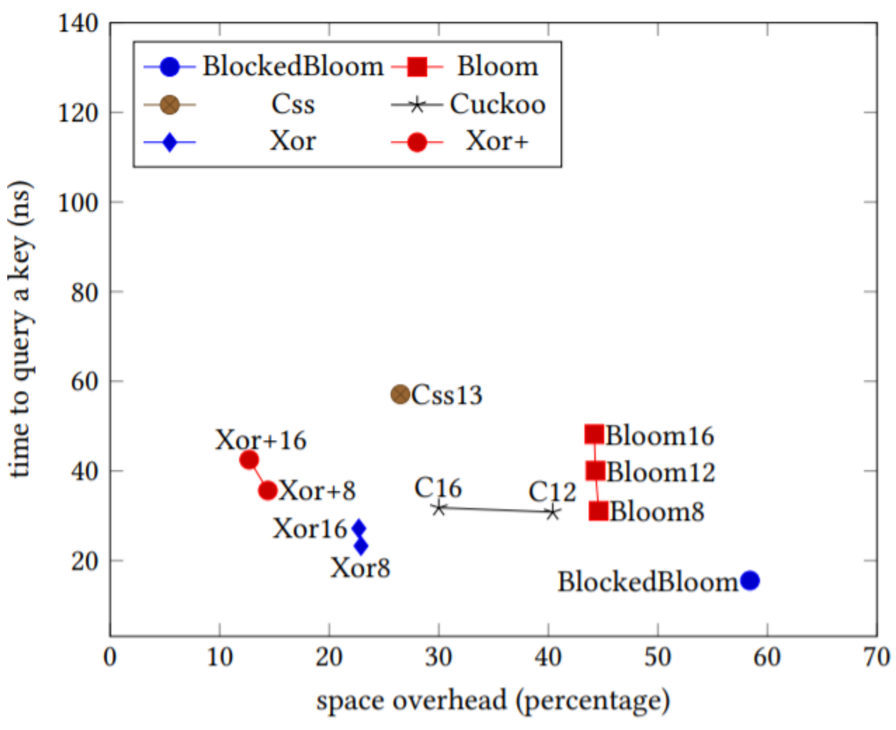
XORFilter

- Зачем надо?
  - Если надо ещё поулучшить Bloom!
  - Для (относительно) статических данных
  - Поменьше памяти, на **~15% меньше** теор-оптимум
  - Побыстрее работает, **~25% быстрее**
- Чем плохо?
  - Медленнее генерируется
  - Неинкрементально обновляется



XORFilter





(a) 10M keys



XorFilter

```
// prebuilt, read-only data for querying
struct XorFilter {
  int n = 100;
  int fp, h0, h1, h2; // "magic" hashfunc seeds
  char table[155]; // 32 + 1.23 * n

  bool IsThere(Object & val) {
     char t0 = table[hash(val, h0)];
     char t1 = table[hash(val, h1)];
     char t2 = table[hash(val, h2)];
     return hash(val, fp) == t0 ^ t1 ^ t2;
  }
```

XORFilter

- Почему работает?!
  – Работает, тк. h0, h1, h2 “ловко подбираются”
  – Быстрый, тк. меньше (!) вычислений и mem reads
- Где взять?
  – https://github.com/FastFilter/
  – (куда ни плюнь, кругом Daniel Lemire, ага)
  – Есть для C++, Java, Golang, Rust, Erlang итд итп



продолжим по вероятностям

T-Digest

T-Digest

- Зачем надо?
  - Быстрая компактная оценка квантилей
  - “Мы льем 10K rps, каковы p50, p99, p87?”

  - Сохранять *весь* поток, понятно, неохота!!!
  - И даже уник-значений мб ***много***, eg. latency
  - (Таблица 1-1000 msec и “хвост” впрочем…)



T-Digest

7.×10⁻⁶
6.×10⁻⁶
5.×10⁻⁶
4.×10⁻⁶
3.×10⁻⁶
2.×10⁻⁶
1.×10⁻⁶
0

200 000
400 000
600 000
800 000



T-Digest

```
struct Centroid {
  float mean, weight;
};

struct TDigest
{
  vector<Centroid> centroids; // just 200 is quite good!
  int max_size;
  double sum = 0.0, count = 0.0;
  double max = NAN, min = NAN;
};
```

T-Digest

- Почему работает?
  - Сохраняем “огрубленный” график DF
  - Сохраняем только “интересные” местам
  - Это и есть т.н. “центроиды”
  - …И как обычно, батчим вставки (скорость!)



T-Digest

- Чем плохо?
  - Может слегка промазать
- Где взять?
  - https://github.com/tdunning/t-digest
  - …там же и почитать AddBatch(), GetQuantile()
  - …увы, там по 70-100 строк ☺



HyperLogLog

HyperLogLog

- Зачем надо?
  - компактная поточная оценка “кардинальности”
  - “нам влетает 1B штук, хотим approx_count_distinct()”
  - штука == db_row, uniq_client_IP, {conn_src + conn_dst}…
- Но как?! (если без хранения-то?!)
  - Магия математики!!!
  - **Раз,** N *уникальных* uniform значений из {0,1}^inf
  - Должны увидеть ~N/2^K префиксов 000…1 длиной K
  - **Два,** хэши унд бакеты



HyperLogLog

```
struct HyperLogLog {
  static const int BITS = 12; // bits
  static const int REGS = (1 << b); // ie. 64 "registers"
  char regs[REGS] = {0}; // {0} means "all zeroes"

  void Add(Object & val) {
    uint h = hash(val);
    int j = h >> (32 - BITS); // ie. 26
    int r = LeftmostBit(h << BITS); // 1 based, ie. 1+clz()
    regs[j] = max(regs[j], r);
  }
```




HyperLogLog

```
struct HyperLogLog {
  static const int BITS = 12; // bits
  static const int REGS = (1 << b); // ie. 64 "registers"
  char regs[REGS] = {0}; // {0} means "all zeroes"

  int64_t GetCount() {
    double sum = 0;
    for (auto v : regs)
      sum += 1 / pow(2, v);
    return 0.79402 * REGS * REGS / sum;
  }
```




HyperLogLog

- Почему работает?!
  - “потому что математика”
  - Либо оригинальные (и новые!) статьи, Flajolet et al
  - Либо “Кое-что о вероятностных СД”, Ajtkulov, HL’2019
- Чем плохо?
  - Точности может не хватить (ну те. запросто 2 знач-разряда)
  - Поэтому extreme care
- Где взять?
  - Выглядит, что написать (даштоштакое)



…бдыщь, “у Стэнфорда” была ***вторая*** структура!!!

SkipList

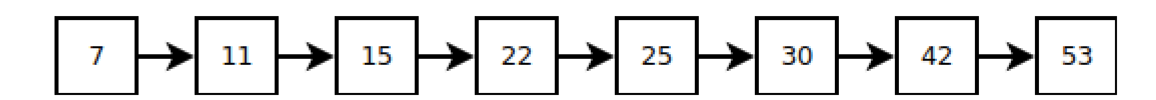


SkipList

- Когда надо?
  - Когда надо **“как sorted array/list”**, но лучше!
  - Дешевый sorted walk, иначе зачем всё
  - Дешевый Find(), как в sorted array
    - O(log n), дешевле только *unsorted* хэш
  - Дешевые Add()/Remove()
    - O(log n) вместо O(n) sorted array, O(1) linked list
- Основная идея?
  - Linked list базово
  - Рандомизированные (!!!) ”прореженные” lists сверху
  - Получается BST без “честного” ребала; в среднем сходится

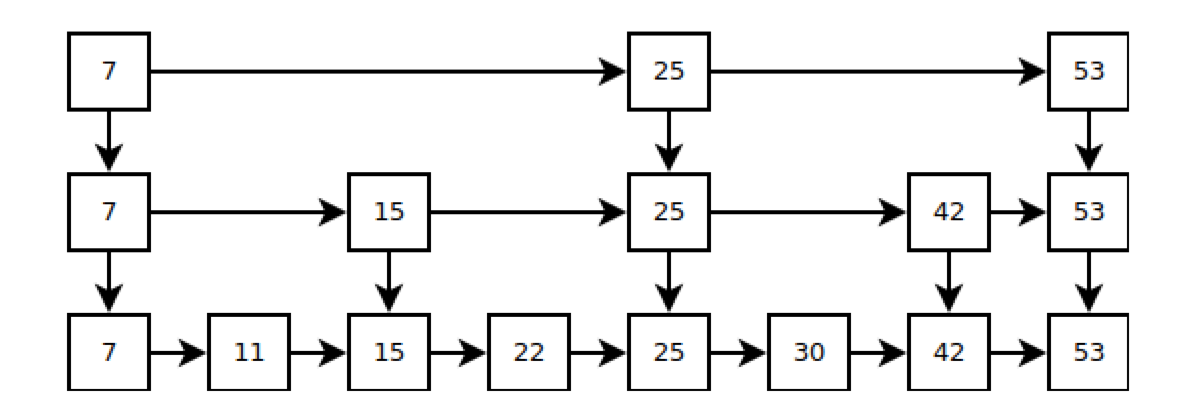


SkipList





SkipList





SkipList

7 25 53

7 15 25 42 53

7 11 15 22 25 30 42 53



SkipList

- По-русски говори, когда надо?!
  - например… а я не очень знаю, честно говоря ;(
  - очень похоже на binary tree, то есть std::map, нооо…
  - говорят (в википедии), Redis ordered sets, MemSQL, etc
  - говорят, для **threaded** (sic) **ordered** (sic) **maps** жжот!
  - **“10x faster than a RWSpinLocked std::set for 1K-1M nodes”**
- Где взять?!
  - В Java вроде встроено
  - В C++ обратно Folly



SkipList, BST, (NotSo)ScaryTrees… **indexes!**

Группа **индексных** структур

? std::map (aka red-black BST)
✔ SkipList
– Trie
– B-Tree
– LSM

- всё это тн. “индексы”
- Надо и K=>V и range
- Надо и read и write
- ...Обсудим – или сразу завяжем? ☺



Trie



Trie

- Когда надо?
  - K=>V индексы (!) с неким особым балансом
  - Довольно быстрый Find(), рвет многие хэши!
  - Довольно простые в реализации
- Основные идеи?
  - Представим, что ключ это строка (нередко!)
  - Или “а давайте по 26* детишек, не 2 а-ля BST”
  - Или “а давайте схлопывать уровни дерева”



Trie

```
struct TrieNode1 { // sketch
  TrieNode1 *kids[26]; // because A-Z
  int value = MAGIC_NONE; // because 0 is not NULL
};


struct TrieNode2 { // sketch
  TrieNode2 *kids[26]; // because A-Z
  int values[26]; // init with NONEs
};
```




Trie

```
struct TrieNode3 { // sketch
  vector<pair<char,TrieNode3*>> data; // not A-Z
  vector<pair<char,int>> values; // not (!) in sync
};

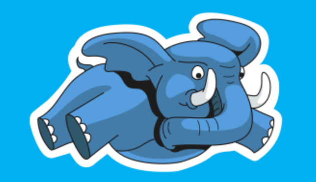

struct TrieNodeX {
  // ...whatever works for you
};
```




Trie

- Чем плохо?
  - Память жрет пушечно!!!
  - O(…) показатели неплохие, но
  - Правильные хэши ещё быстрее, тк. кэш
- Где взять?
  - Подозреваю, лучше написать (охxx)
  - Больно уж кастомные случаи, увы





B-Tree

B-Tree

- Когда надо?
  - K=>V индексы (!) с неким особым балансом
  - Когда собираемся дисковать (и есть кластера)
- Основная идея?
  - ***Очень*** ветвящееся дерево
  - Минимизируем глубину поиска (тк. seek)



B-Tree

```
typedef char Key[13]; // because fu.. i mean we can
typedef float Value;

struct BtreeLeaf {
  int page_type = LEAF; // header magic!
  int used = 0;
  std::pair<Key,Value> data[481]; // 481 is (8192-8)/(13+4)
};

struct BtreeNode {
  int page_type = NODE; // header magic!
  int used = 0;
  std::pair<Key,off_t> kids[389]; // 389 is (8192-8)/(13+8)
};
```




B-Tree

- Чем плохо?
  - Waaagh! Write amplification + seeks
- Где взять?
  - BDB (aka Oracle BerkeleyDB)
  - LMDB
  - https://github.com/google/btree
  - …
  - Писать *боевое* **НЕ** стОит (waaagh)



LSM

LSM

- Когда надо?
  - K=>V индексы (!) с неким особым балансом
  - Когда много вставок (writes) и мало остального
  - Логи и прочие скорее-архивы
  - Типично тоже на диске, но тоже необяз
- Основная идея?
  - Батчинг совсем мелочи + sorted runs + linear merges
  - 1 операция = без апдейта дерева на *каждый* чих
  - 1 вставка = чаще всего дешманский in-mem append!



LSM

```
// say, upto 1K rows in RAM
struct L1Chunk {
  vector<pair<Key,Value>> data; // even unsorted mb ok!
};

// say, upto 32K rows in RAM
struct L2Chunk {
  int n = 0; // upto 32K; and should be within Array
  SortedArray<pair<Key,Value>> data;
  Bitmap killed;
};
```

LSM

```
// say, upto 1M rows chunk on disk
struct L3ChunkDisk : L2Chunk, L3Meta {
  // ...
};

// say, always-in-RAM info for that L3 chunk!
struct L3Meta {
  BloomFilter keys_filter;
  vector<pair<Key,off_t>> key_blocks;
};
```




omfg, it’s all… **connected**





LSM

- Чем плохо?
  - При перекосе в поиск мб-не-очень
  - Рискуем сходить в отн. много sorted runs
  - Как обычно, bench bench bench
- Где взять?
  - Писать явно неохота :)
  - https://github.com/facebook/rocksdb
  - https://github.com/google/leveldb
  - …и как минимум почитать про Vinyl статьи



это метамаркер (я не планировал сюда успеть)

(сейчас вы спросите)

TENET!

вопросы!



are you even shpongled, bruv?



В чем суть!?

- Жизнь она интересная
- Жизнь **есть** и кроме vector, hash, map/list

- Вот, мы заглянули одним глазом за забор
- Вот, если (если) вам потребуется, то вы готовы! :)
- Вот, они ж ***все**** прям несложные, скетчи по 3-10 LOC
  - иначе есть либа, даже от целых F или G

- Не не не, точно НЕ потребуется никогда? Штош
- Не не не, даже ни половинки ни мелкой идеи? Штош



Что ***эзотерического*** осталось?!

- О, еще *куда* больше!
  - y-fast trees // approx ordered sets // succinct lists // patricia tries // van emde boas layout & trees // select heap // cuckoo hashing // ropes // ctries // minhash // splay trees // roaring bitmaps // HAT tries // interval trees // …
- Но вам (и нам) туда скорее не надо
- Там живут скорее закшвар-драконы-академики ☺
- Тут таки ***прикладная*** эзотерика ☺



Что ***прикладного*** осталось?!

- “Нечастые” (**непоголовные!**) СД
- Heap // BRIN // …

- Нечастые А
- Quickselect // Perfect hashing // tANS // kNN indexes // …

- Нечастые турбо-реализации всякого
- Ryu // simdJSON // roaring // …

- …Путь, как обычно, бесконечен (пока не закончится)



Нечастые “странные” ***задачи***

• B-Tree	disk index
• Bitmap	limited range set
• BloomFilter	presence estimate
• HyperLogLog	cardinality estimate
• LSM	disk index
• SkipList	concurrent RAM index
• SparseSet	limited range set
• T-Digest	percentile estimate
• Trie	RAM index
• XORFilter	presence estimate

  – Таки 5-6 разных задач (смотря как зачесть индекс)



Что пробежали ***прикладное***?!

- B-Tree
- Bitmap
- BloomFilter
- HyperLogLog
- LSM
- SkipList
- SparseSet
- T-Digest
- Trie
- XORFilter
  - Даже если у всех 10/10, я считаю, всё равно успех!



…это ещё не конец (с)

TENET!